AN1256

Interfacing the HC05 MCU to a Multichannel Digital-to-Analog Converter Using the MC68HC705C8A and the MC68HC705J1A

By Mark Glenewinkel
CSIC Applications
Austin, Texas

INTRODUCTION

This application note describes the interface between Motorola's HC05 Family of microcontrollers and Maxim's MAX528/MAX529 (529) digital-to-analog converter (DAC). The 529 is an 8-bit, 8-channel, serial interface DAC with programmable output buffers. The microcontroller unit (MCU) interface must be able to "talk" to the 529 using a serial communication link. The serial peripheral interface (SPI) is one of the most widely used serial transmission methods for communication between an MCU and a peripheral. This application note describes the hardware and software design needed to link the SPI module on the MC68HC705C8A MCU to the 529.

Not all HC05 Family members have SPI modules. An HC05 MCU without an SPI must interface with the 529 using a software input/output (I/O) driver. This method uses software bit programming to communicate with the 529. Although not as efficient as the hardware SPI method, it provides MCUs with a means to send data to the 529. This application note utilizes the MC68HC705J1A MCU to demonstrate the software driver routine.

MAX529 DIGITAL-TO-ANALOG CONVERTER

Overview

The MAX529 is a monolithic device consisting of eight voltage output DACs. Two reference voltage inputs feed two sets of four DACs on the chip. A serial interface is used to communicate with the chip. The MAX528 operates from split supplies totaling up to 20 V, including +5 V and -15 V, +12 V and -5 V, and +15 V and -5 V, or a single supply up to 15 V. The MAX529 operates from 5 V supplies or from a single +5 V supply. This application note utilizes the MAX529 with a single +5 V supply. If low-power consumption is required, the part can be put in shutdown mode with its shutdown pin. During shutdown, the part uses less than 50 A of current.

The part can configure its buffer mode of the DAC output pins in three different ways:

1) An unbuffered mode connects the internal R-2R DAC network directly to the output pin.
2) A full-buffered mode inserts an op amp buffer between the R-2R network and the output pin, providing a +5 mA and -2 mA output drive.
3) A half-buffered mode is similar to the full-buffered mode but only provides up to +5 mA of output drive in a unipolar configuration.

If needed, the part can be serially daisy-chained to other 529s to increase the number of DACs in a system.

© Motorola, Inc., 1995

The R-2R DAC Network

The DAC inside the MAX529 is based on the R-2R resistor network. Most CMOS DACs are based on the R-2R current steering circuit. Figure 1 shows a simple 2-bit R-2R DAC. A reference voltage is applied to the V_{REF} pin and the current, I, is binarily divided throughout the array as shown. These currents are steered in discrete incremental amounts to the OUT1 and OUT2 nodes. The digital input to the DAC determines the position of the switches used to steer the current. A logic one causes the switch to steer the current to OUT1, while a logic zero causes the switch to steer the current to OUT2. OUT2 is at analog ground. The feedback configuration of the op amp forces OUT1 to be at virtual ground potential.

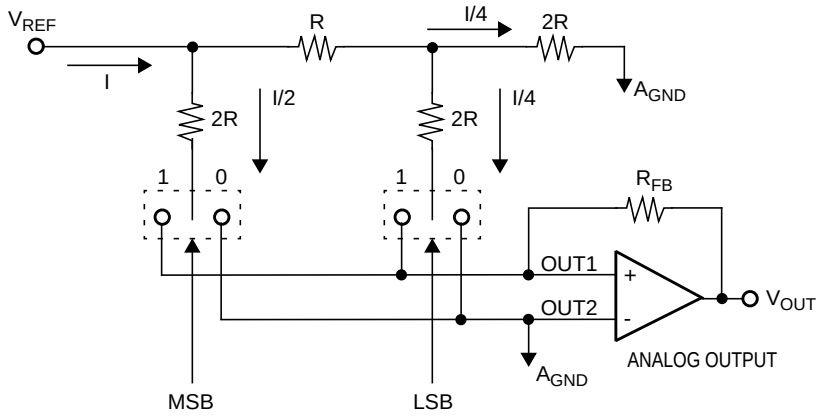


Figure 1. Simple 2-Bit Digital-to-Analog Converter

In this example, a digital value of 10_2 causes I/2 to flow to OUT1 and the remainder of the current flows to OUT2. Therefore, 10_2 refers to half scale. If the input to the DAC was 11_2, the output current would be full scale minus one LSB. In this example, the full-scale current reading would be 3/4I.

The R-2R DAC will perform only if the OUT1 and OUT2 nodes are at the same potential. Therefore, a current-to-voltage op amp converter is used. The feedback resistor R_{FB} is made equal to R. The maximum output voltage for this configuration is $-I(1-2^{-n})R$ where n is the number of bits of DAC resolution. The minus sign in the output voltage is a result of the current-to-voltage conversion. Another inverting op amp buffer with gain of -1 may be used to create a positive output voltage. The resultant voltage output of the DAC then can be defined as follows:

$$V_{OUT} = V_{REF} * (xx/2^n)$$

where:

xx is the digital input to the DAC and n is the bit resolution of the DAC

For the 2-bit DAC above, the available output voltages are 0 V_{REF}, 1/4 V_{REF}, 1/2 V_{REF}, and 3/4 V_{REF}.

Inside the MAX529

As stated earlier, the 529 contains eight latched 8-bit DACs, eight buffer amplifiers, serial control logic, and two reference inputs. The buffer amplifiers can be configured as buffered, half buffered, and unbuffered. With one 16-bit serial transmission, any or all of the eight voltage outputs can be programmed. Figure 2 shows the block diagram of the 529.

The 529 DACs are divided into two groups of four DACs. Each group has its own REFH and REFL analog input reference voltages. The output of a DAC is defined as

$$V_{OUT} = (REFH-REFL) * (xx/256 + REFL)$$

where:

xx is the 8-bit digital input code with a range of 0-255.

Unbuffered mode connects the internal R-2R DAC network directly to the output pin. Full-buffered and half-buffered modes allow the user to drive more of a load directly on the outputs of the 529. All electrical specs for operating voltages, reference voltages, and buffer modes can be found in greater detail in the 529 data sheet.

Digital Interface

The digital interface to the 529 is composed of a serial data port that synchronously transmits 16-bit data. It also is capable of being shut down by an external pin to conserve power.

$\overline{CS}$ Active-Low Chip Select

When asserted low, this input pin initializes the 529 to start a new frame of serial data. When asserted high, the 16-bit data is latched and the internal shift register is turned off. The DAC registers also are updated with the new data.

DOUT Serial Data Out

This open drain pin serves as the serial output data from the DIN pin.

DIN Serial Data In

This pin serves as the input data line that receives the 16-bit serial data stream.

CLK Serial Data Clock

This pin is an input that drives the serial transmission lines.

$\overline{SHDN}$ Shutdown

Connect this input pin high for normal operation. Connect it low to conserve power.

Serial data is clocked in at DIN on the rising edge of CLK after $\overline{CS}$ is asserted low. Refer to Figure 3. After all 16 bits have been clocked in, the $\overline{CS}$ pin is negated to latch the data. The DAC outputs and buffers will be changed according to the latched data. The serial output DOUT pin is an open-drain FET that requires a pullup resistor (typically 4.7 KΩ) to V_{DD}. Any number of 529s can be daisy-chained together by connecting the DOUT pin of one device to the DIN pin of the following device in the chain.

DAC Programming

The 529 is programmed with 16 bits of information. The first eight bits contain the address pointer and the second eight bits contain the data byte. These bits enter a shift register serially through DIN with A7 first and D0 last.

Setting the DAC Outputs

To program one of the eight DACs, the corresponding bit in the address pointer must be set and the data byte must hold the digital data to set the correct voltage for that output. Any or all of the outputs may be set according to the address pointer. This instruction will change only the outputs, not the buffers.

Setting the Buffers

To set the buffers, all address pointer bits must be set to zero and data bit D7 is one. When this instruction is sent to the 529, data bit D6 is ignored and D5-D0 is latched into the mode registers only. The DAC registers are unaffected. Refer to Table 1 for programming the buffers.

Table 1. Buffer Mode Programming

Mode	OUT0,1	OUT2,3	OUT4,5	OUT6,7
Unbuffered (D0 = D3 = X)	D5 = 0	D4 = 0	D2 = 0	D1 = 0
Half-Buffered	D5 = 1	D4 = 1	D2 = 1	D1 = 1
	D3 = 0		D0 = 0	
Full-Buffered	D5 = 1	D4 = 1	D2 = 1	D1 = 1
	D3 = 1		D0 = 1	

Programming for Multiple 529s

When programming other 529s configured in a daisy-chain arrangement, use the no operation instruction (NOP) as a place setter. NOP is implemented when all address pointer bits and all data bits are set to zero. When latched into the 529, all outputs and buffers are unaffected.

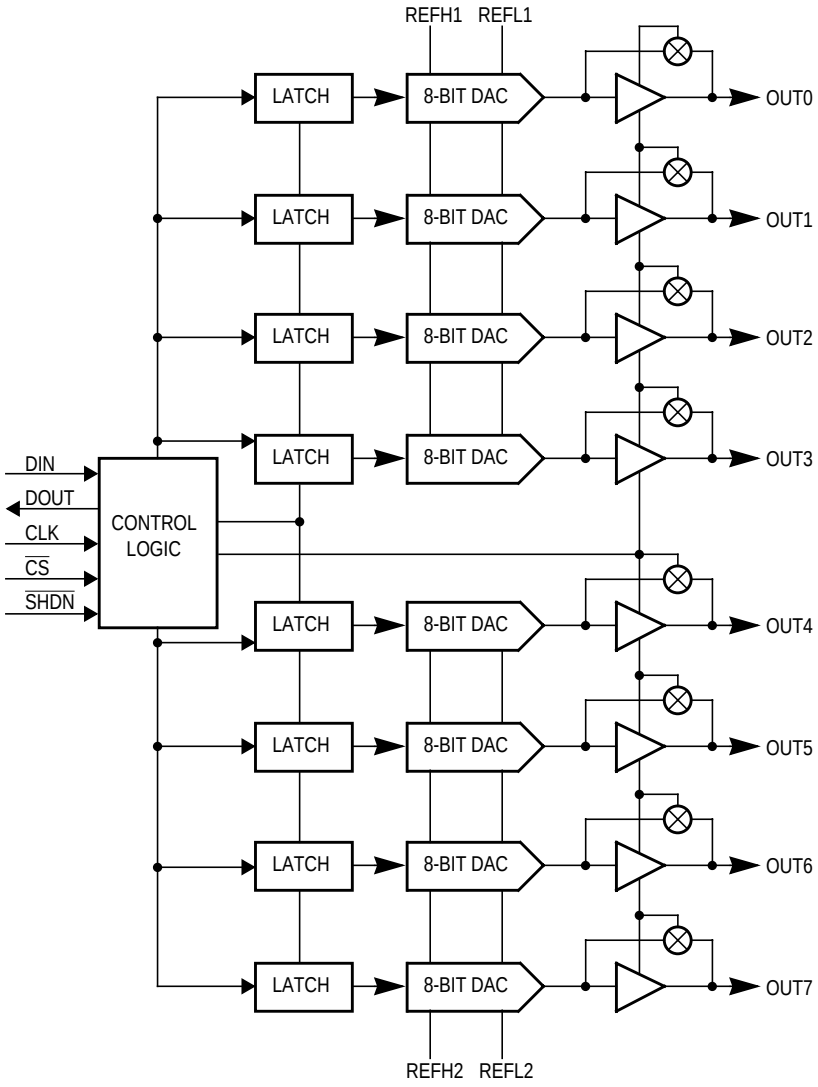


Figure 2. MAX528/529 Block Diagram

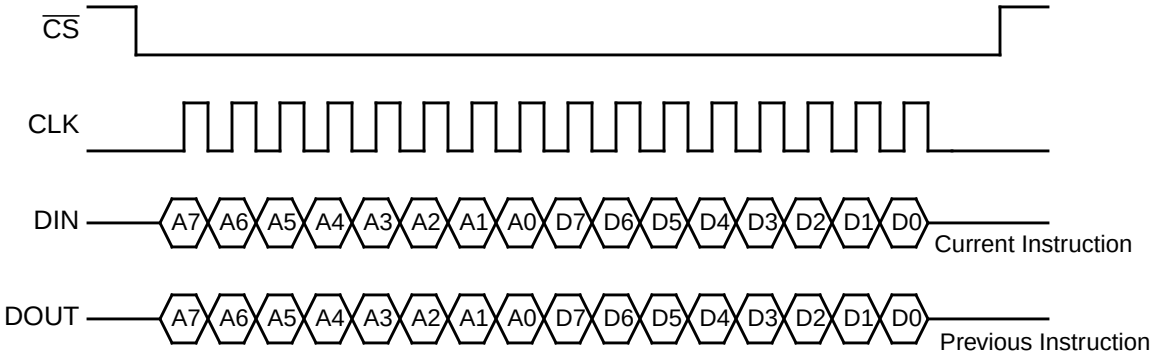


Figure 3. MAX529 Timing Diagram

DESCRIPTION OF THE HC705C8A INTERFACE

Hardware

The MC68HC705C8A is one of the most popular members of the HC05 Family of 8-bit MCUs. It has the serial peripheral interface (SPI) that will be used to interface to the 529. In essence, the SPI is an 8-bit serial shift register that can be manipulated by software instructions. The SPI can be programmed with different clock polarities and clock phases to communicate correctly with a number of devices. The SPI also can be configured to act as a master or a slave. Each SPI signal is explained below. For more detail on the SPI, consult the *MC68HC705C8 Technical Data*, Rev. 1 (MC68HC705C8/D).

SCK Serial Data Clock

The SCK signal is used to synchronize the movement of data in and out of the SPI module. This pin is an output or an input depending on whether the SPI is configured as a master or a slave. Data is shifted on one side of the clock edge and sampled on the other. The SCK signal can be configured to accommodate different serial peripheral bus structures.

MOSI Master Output, Slave Input

When the SPI is configured as a master, this pin is used as an output to shift the 8-bit serial data out with the most significant bit first. The pin is used as a slave data input when the SPI is configured as a slave.

MISO Master Input, Slave Output

If the SPI is configured as a master, this pin is utilized as an input. When the SPI is in slave mode, the pin is used as an output.

$\overline{SS}$ Slave Select

When the SPI is a slave, this pin enables the SPI for an incoming transfer. As a master, this pin should be tied high.

To correctly interface with the 529, the SPI is configured as a master according to the timing diagram shown in Figure 4. This configuration enables the SCK to drive out data with the MOSI pin on the rising edge of the clock.

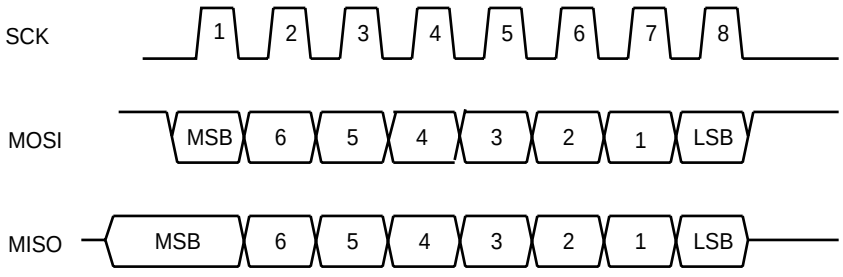


Figure 4. SPI Timing Diagram

The schematic used for this interface is shown in Appendix A. The HC705C8A is clocked by a 4-MHz crystal circuit. This provides the MCU with a 2-MHz internal bus frequency and a 500-ns bus period or instruction cycle. The MC34064 is used as a low-voltage inhibitor circuit. This 3-pin, T0-92 device ensures that the reset pin is pulled low if the operating voltage to the MCU falls below 4.6 volts. All input-only pins are pulled high so that they do not float the internal CMOS gates of the HC705C8A.

The SPI lines are connected to the appropriate pins on the 529. The MOSI pin drives data out of the HC705C8A and into the DIN pin of the 529. Since the SPI is configured as a master, the SCK pin of the HC705C8A drives the CLK pin of the 529 and the $\overline{SS}$ pin is tied high.

The 529 is configured in a single supply operation. The output DAC voltages range between 0 and 2 volts. This is accomplished by the voltage divider circuit hooked to the V_{DDA} analog power supply of 5 volts. This 2-volt reference voltage is fed into pins REFH1 and REFH2. V_{DD} is connected to the $\overline{SHDN}$ pin for normal operation.

The circuit given in Appendix A minimizes the noise often found in emulated systems. It will only work if code has been burned into the HC705C8A's internal EPROM. Instead of programming the HC705C8A, the MMDS05 can be used to emulate the HC705C8A code. This method allows more flexibility in code development than using a programmed HC705C8A.

Test Software

The flowchart for the SPI-driven 529 is shown in Appendix B, and the actual HC05 assembly code is given in Appendix C. This code was written for a programmed HC705C8A. Extra lines of code were added so that the routine will perform in a standalone application.

For the SPI to "talk" to the 529, the SPI must be configured to match the 529 timing diagram, as shown in Figure 3. Also, two SPI transmissions must be sent to form a 16-bit transfer. Before any transmissions can start, the $\overline{CS}$ pin must be asserted low. This initializes the 529 and tells it that a new address and data will be sent to it to start the conversion process. The first transfer sends the address pointer to the 529. The second transfer sends the DAC data byte to the 529. After both transmissions are completed, $\overline{CS}$ is negated high and the DAC outputs are updated. This transmission sequence has been put into a subroutine called CM529_TXD.

The main routine in Appendix C will output a waveform on all DAC outputs. The waveform is a stairstep function with a frequency of 2.336 kHz. It is diagrammed in Figure 5.

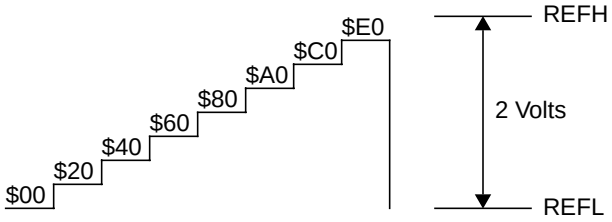


Figure 5. DAC Output Stairstep Function

The instructions below are provided to test the software routine. Follow these steps after programming the HC705C8A with the code in Appendix C and constructing the schematic in Appendix A.

1) Power-on the circuit.
2) Check that the oscillator circuit on pin 38 of the HC705C8A is running at 4 MHz.
3) Verify that the $\overline{RESET}$ pin on the HC705C8A is 5 volts.
4) Verify that pins REFH1 and REFH2 on the 529 are at 2 volts.
5) Pins OUT0 to OUT7 on the 529 should be driving out the waveform shown in Figure 5 at a frequency of 2.336 kHz.

DESCRIPTION OF THE HC705J1A INTERFACE

Hardware

With only 20 pins, the HC705J1A is one of the smaller members of the HC05 Family. It has a total of 1240 bytes of erasable programmable read-only memory (EPROM) and includes 14 I/O pins. The schematic for the HC705J1A-to-529 interface is shown in Appendix D. The circuitry surrounding the 529 is the same as in the HC705C8A design. The only changes are the serial pins of the 529. These pins are connected to two I/O pins on the HC705J1A. The pins used to drive the 529 on the HC705J1A are as follows:

Port A, Bit 0 —This I/O pin ($\overline{CS}$) is configured as an output to drive the $\overline{CS}$ pin on the 529.

Port A, Bit 1 —This I/O pin (SER_CLK) is configured as an output to drive the serial clock of the serial transmission bus.

Port A, Bit 2 —This I/O pin (SER_OUT) is configured as an output to drive the serial data out and into the DIN pin of the 529.

For further information on the HC705J1A, consult the *MC68HC705J1A Technical Data* (MC68HC705J1A/D).

Test Software

I/O manipulation is the process of toggling I/O pins with software instructions to emulate a certain piece of hardware peripheral. The flowchart for the I/O-driven 529 is shown in Appendix E, and the actual HC05 assembly code is given in Appendix F. This routine was written especially for the 529 and is not a full-featured representation of the HC705C8A SPI module. Enhancements to the routine were not included to maximize the code's efficiency.

As stated in the preceding hardware section, I/O pins have been used to send out the correct serial transmission protocol to the 529. The HC05 CPU provides special instructions to specifically manipulate single I/O pins. The 529 serial stream shown in Figure 3 will be re-created by three I/O pins on the HC705J1A.

This transmission has been put into a subroutine called CM529_TXD. The best way to describe the subroutine is to list each segment of the code to explain the I/O during transmission. For example:

Initialization
Clear the CS pin
Load the X register with 16; use it as a counter

Write the serial output pin
Bit 7 of DAC_ADDR is read. If it is high, a one is written to SER_OUT. If it is low, a zero is written to SER_OUT.

Clock the serial clock pin
The SER_CLK pin is written high and then written low.

Rotate the data bytes
Rotate left the DAC_DATA byte
Rotate left the DAC_ADDR byte
Decrement the X register

Is the loop done?
The index register is decremented and checked to see if it is zero. If X is not zero, the code is executed at the start of writing the SER_OUT pin. This loop continues until 16 transmissions are completed. When finished, a return from subroutine is executed.

The main routine in Appendix F will output a waveform on all outputs of the DAC. The waveform is a stairstep function with a frequency of 382 Hz. It is diagrammed in Figure 5.

The instructions below are provided to test the software routine. Follow these steps after programming the HC705J1A with the code in Appendix F and constructing the schematic in Appendix D.

1) Power-on the circuit.
2) Check that the oscillator circuit on pin 2 of the HC705J1A is running at 4 MHz.
3) Verify that the $\overline{\text{RESET}}$ pin on the HC705J1A is 5 volts.
4) Verify that pins REFH1 and REFH2 on the 529 are at 2 volts.
5) Pins OUT0 to OUT7 on the 529 should be driving out the waveform shown in Figure 5 at a frequency of 382 Hz.

LAYOUT CONSIDERATIONS

Many considerations apply when laying out mixed signal designs such as the 529 and the HC05 MCU. The accuracy of the 529 may be greatly affected if proper layout design is not followed. Listed below are some things to check to ensure the accuracy of the DAC.

- Physically separate critical analog circuits from the digital circuits of the MCU. If possible, split the board in half to separate analog and digital circuits. Each half will have its own power and ground system.
- If at all possible, do not let analog input line traces cross digital traces. But if this must happen, make sure they cross at right angles to each other.
- Use power or ground traces to isolate the analog-input pins from the digital pins.
- With quality ceramic capacitors, bypass the power supplies to the proper ground at the 529 power pins. Keep the bypass capacitors lead lengths as short as possible.
- To bypass low-frequency power supply noise, use tantalum electrolytic capacitors of 5 to 20 μF. These should be placed near the point the power supplies enter the board.
- If economically possible, use separate analog and digital ground planes. The two ground planes should be tied together at the low impedance power-supply source.

REFERENCES / FURTHER READING

Analog-Digital Conversion Handbook, Third Edition, New York: Prentice-Hall, 1986.

MC145050/51 Technical Data Sheet, (MC145050/D), Motorola, 1993.

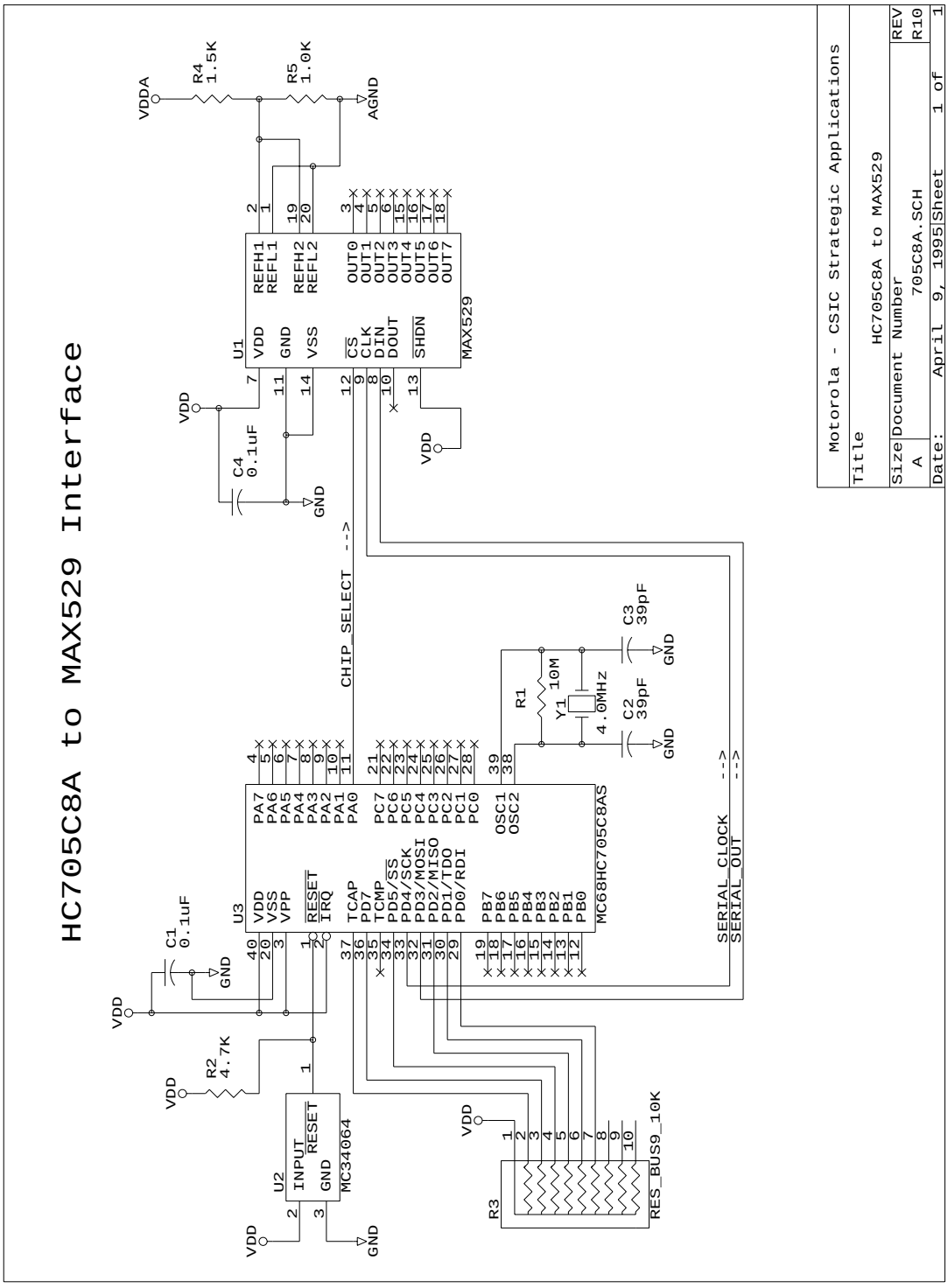
MC68HC05 Applications Guide, (M68HC05AG/AD), Motorola, 1989.

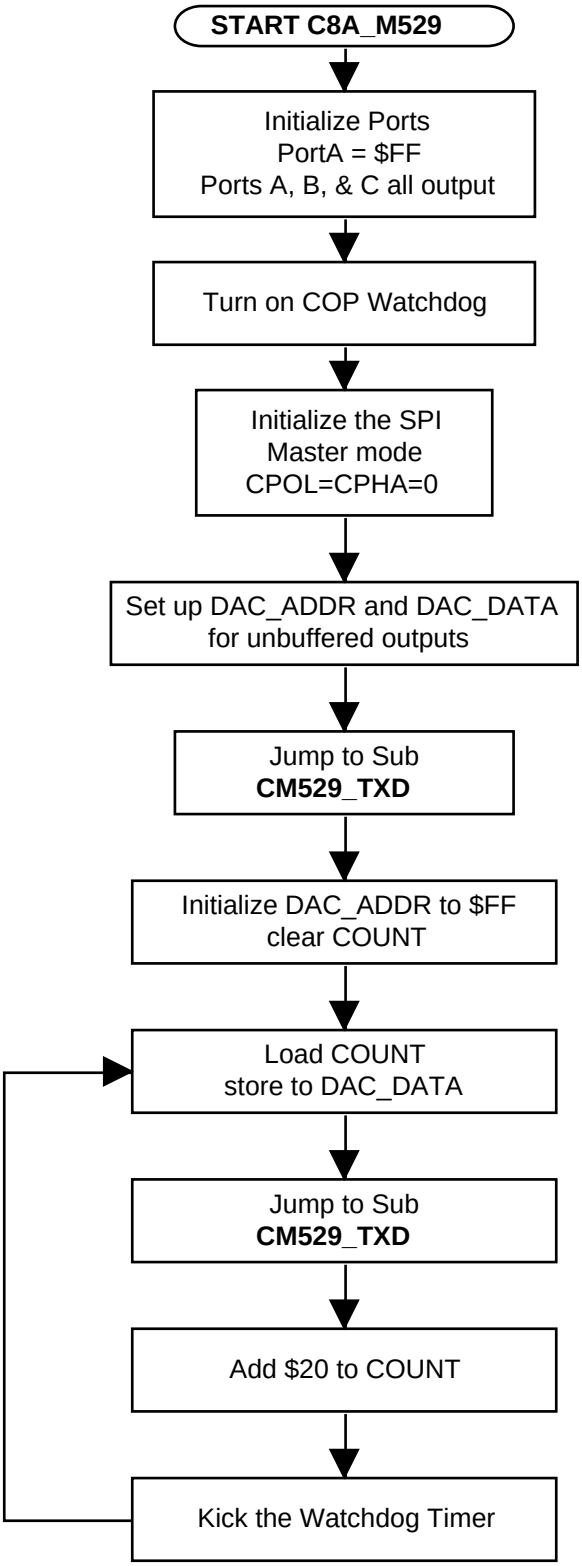
MC68HC705C8 Technical Data, (MC68HC705C8/D Rev. 1), Motorola, 1990.

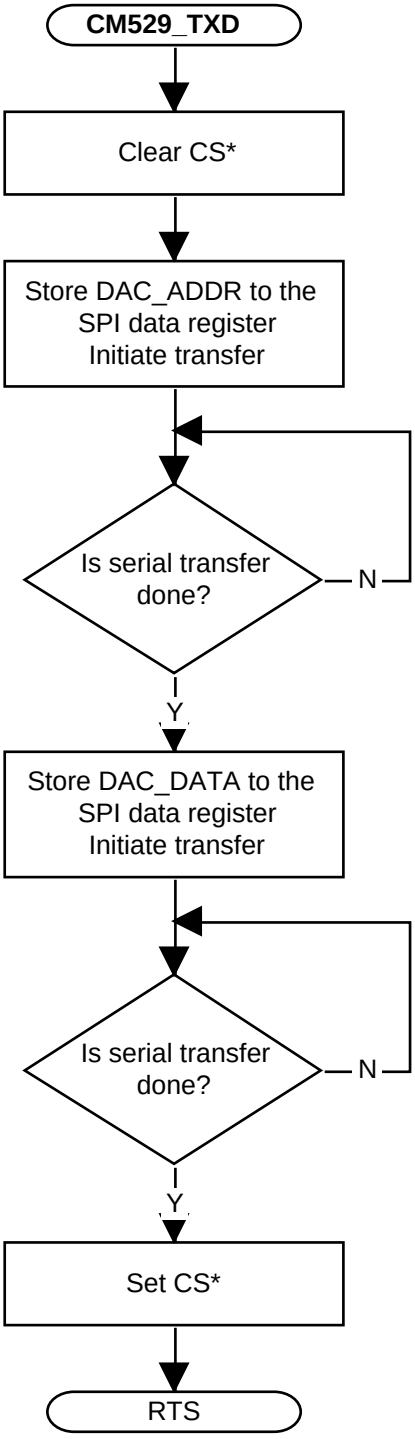
MC68HC705J1A Technical Data, (MC68HC705J1A/D), Motorola, 1995.

Reducing A/D Errors in Microcontroller Applications, (AN1058/D), Motorola, 1990.

MAX528/MAX529 Data Sheet, (19-4193 Rev. 1), Maxim, 1992.


HC705C8A to MAX529 Interface
Motorola - CSIC Strategic Applications
Title
HC705C8A to MAX529
Size
A
Document Number
705C8A.SCH
REV
R10
Date: April 9, 1995
Sheet 1 of 1
VDDA
R4 1.5K
R5 1.0K
AGND
U1
MAX529
REFH1
REFL1
REFH2
REFL2
VDD
GND
VSS
CS
CLK
DIN
DOUT
SHDN
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
C4 0.1uF
CHIP_SELECT -->
U3
MC68HC705C8AS
VDD
VSS
VPP
RESET
IRQ
TCAP
PD7
TCMP
PD5/SS
PD4/SCK
PD3/MOSI
PD2/MISO
PD1/TDO
PD0/RDI
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
PC7
PC6
PC5
PC4
PC3
PC2
PC1
PC0
OSC1
OSC2
PB7
PB6
PB5
PB4
PB3
PB2
PB1
PB0
C1 0.1uF
R1 10M
Y1 4.0MHz
C2 39pF
C3 39pF
SERIAL_CLOCK -->
SERIAL_OUT -->
R2 4.7K
U2
MC34064
INPUT
RESET
R3
RES_BUS9_10K


START C8A_M529
Initialize Ports
PortA = $FF
Ports A, B, & C all output
Turn on COP Watchdog
Initialize the SPI
Master mode
CPOL=CPHA=0
Set up DAC_ADDR and DAC_DATA
for unbuffered outputs
Jump to Sub
CM529_TXD
Initialize DAC_ADDR to $FF
clear COUNT
Load COUNT
store to DAC_DATA
Jump to Sub
CM529_TXD
Add $20 to COUNT
Kick the Watchdog Timer


CM529_TXD
Clear CS*
Store DAC_ADDR to the SPI data register Initiate transfer
Is serial transfer done?
N
Y
Store DAC_DATA to the SPI data register Initiate transfer
Is serial transfer done?
N
Y
Set CS*
RTS

Appendix C
HC705C8A/529 Assembly Code

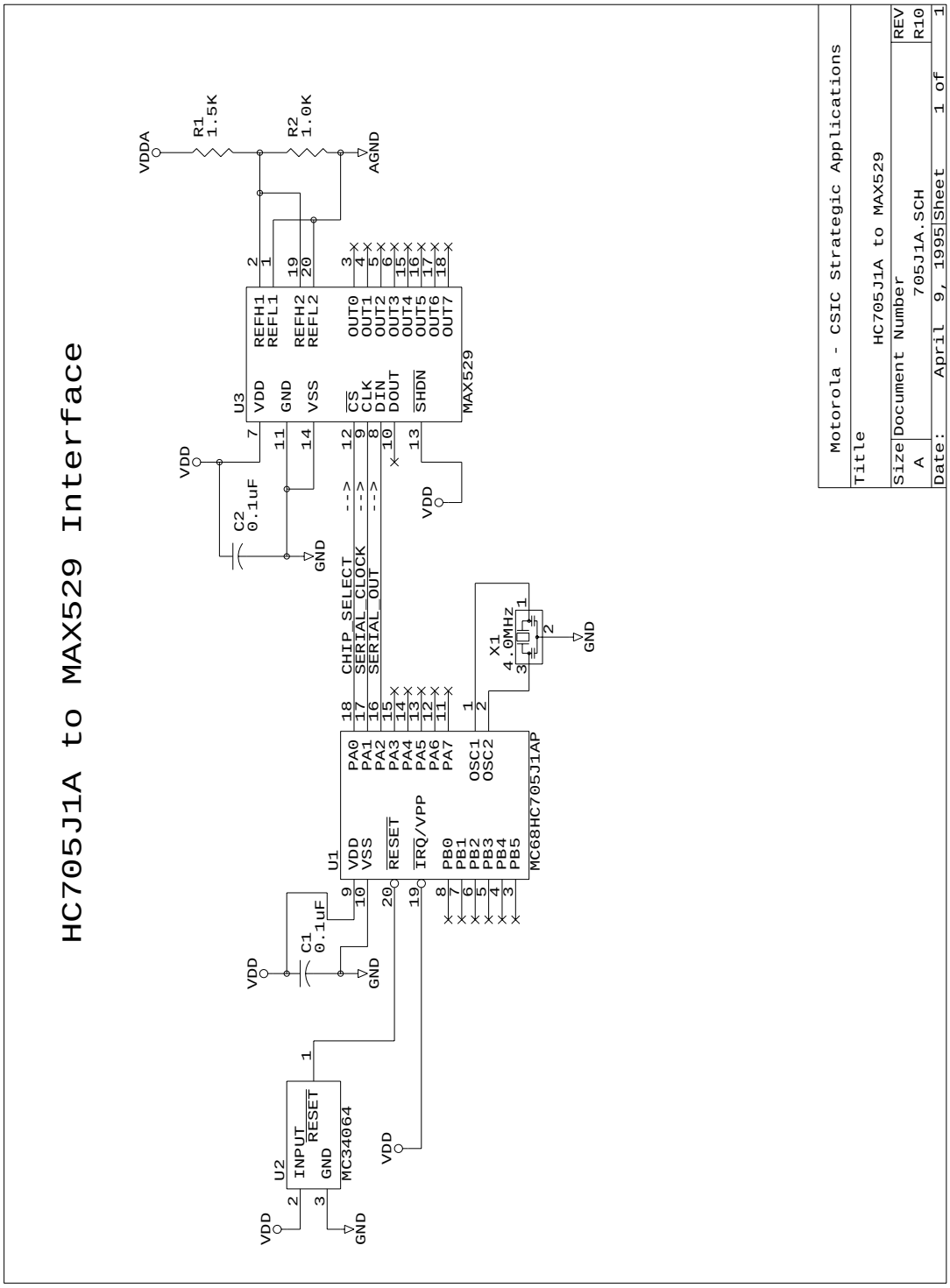
```
*****************************************************************************
*****************************************************************************
*                                                                           *
*            Main Routine C8A_M529 - 705C8A to MAXIM MAX529 DAC             *
*                                                                           *
*****************************************************************************
*                                                                           *
* File Name: C8A_M529.RTN                    Copyright (c) Motorola 1995    *
*                                                                           *
* Full Functional Description Of Routine Design:                            *
*   Program flow:                                                           *
*       Reset:     Initializes ports for bit banging.                       *
*                  Set up MAX529 for half-buffered mode                     *
*                  Initialize DAC address and count for test                *
*                  Execute continuous loop to create stair step function    *
*       CM529_TXD: Clear CS                                                 *
*                  Write address with SPI                                   *
*                  Write data with SPI                                      *
*                  Set CS                                                   *
*                                                                           *
*****************************************************************************
*                                                                           *
*       Part Specific Framework Includes Section                            *
*                                                                           *
* Place the assembler statement (#INCLUDE) to include the part specific     *
* framework for the target part.                                            *
*                                                                           *
*****************************************************************************

#nolist
#INCLUDE 'H705C8A.FRK'                  ;Include the equates for the HC705C8A
                                        ;so that all labels can be used.
#list

*****************************************************************************
*                                                                           *
*       MOR Bytes Definitions for Main Routine                              *
*                                                                           *
*****************************************************************************

                org     MOR1
                db      $00             ;nothing changed

                org     MOR2
                db      $00             ;nothing changed
```

```
*****************************************************************************
*                                                                           *
*       Equates and RAM Storage                                             *
*                                                                           *
*****************************************************************************
CS              equ     0               ;bit # for chip select

***     RAM storage variables   ***

                org     RAM_Start       ;start of static RAM at $50
DAC_ADDR        rmb     1               ;1 byte needed for DAC address
DAC_DATA        rmb     1               ;1 byte needed for DAC data
COUNT           rmb     1               ;1 byte for counting

*****************************************************************************
*                                                                           *
*       Program Initialization                                              *
*                                                                           *
* This section sets up the port for bit banging.                            *
* The COP Watchdog is enabled.                                              *
* The SPI is initialized.                                                   *
*                                                                           *
* To prevent floating inputs and associated high current draw,              *
* the HC705C8A I/O pins have been initialized to outputs.                   *
*                                                                           *
*****************************************************************************

                org     EPROM_Start     ;start of user eprom at $160
CM529_START     lda     #$FF
                sta     PORTA           ;port A = $FF
                sta     DDRA            ;port A all outputs
                sta     PORTB
                sta     DDRB            ;port B all outputs
                sta     PORTC
                sta     DDRC            ;port C all outputs

*       Turn on COP watchdog
                lda     #$04
                sta     COPCR           ;turn on COP

*       Initialize SPI module
                lda     #$50            ;turn on spi, mstr mode
                sta     SPCR            ;cpha=cpol=0
```

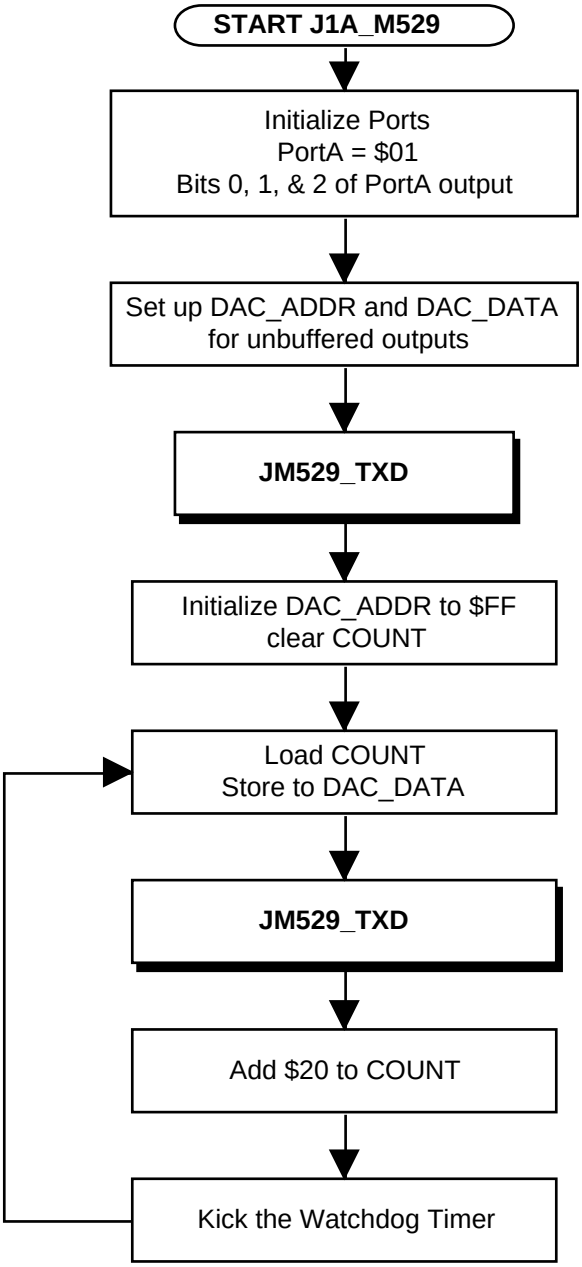
```
*****************************************************************************
*                                                                           *
*       C8A_M529 Main Program Loop                                          *
*                                                                           *
* The code runs through the routine to check for                            *
* proper serial transmission. Each DAC channel will have a 2.336kHz         *
* stair step function                                                       *
*                                                                           *
*****************************************************************************


*       Set up DAC for un-buffered output mode
                lda     #$00            ;load up DAC_ADDR=$00
                sta     DAC_ADDR
                lda     #$80            ;load up DAC_DATA=$80
                sta     DAC_DATA
                jsr     CM529_TXD       ;transmit info to MAX529

*       Initialize DAC address and COUNT for test
                lda     #$FF            ;all DAC channels will be tested
                sta     DAC_ADDR
                clr     COUNT           ;clear COUNT

*       Loop to output stair step function on all 8 DAC outputs
CM529_Loop      lda     COUNT
                sta     DAC_DATA        ;store COUNT to DAC_DATA
                jsr     CM529_TXD       ;transmit info to MAX529

                lda     COUNT           ;add $20 to COUNT
                add     #$20
                sta     COUNT

*       Kick the Watchdog and Loop back
CM529_BRANCH    lda     #$55            ;reset COP
                sta     COPRST
                lda     #$AA
                sta     COPRST
                bra     CM529_Loop      ;branch to Loop
```

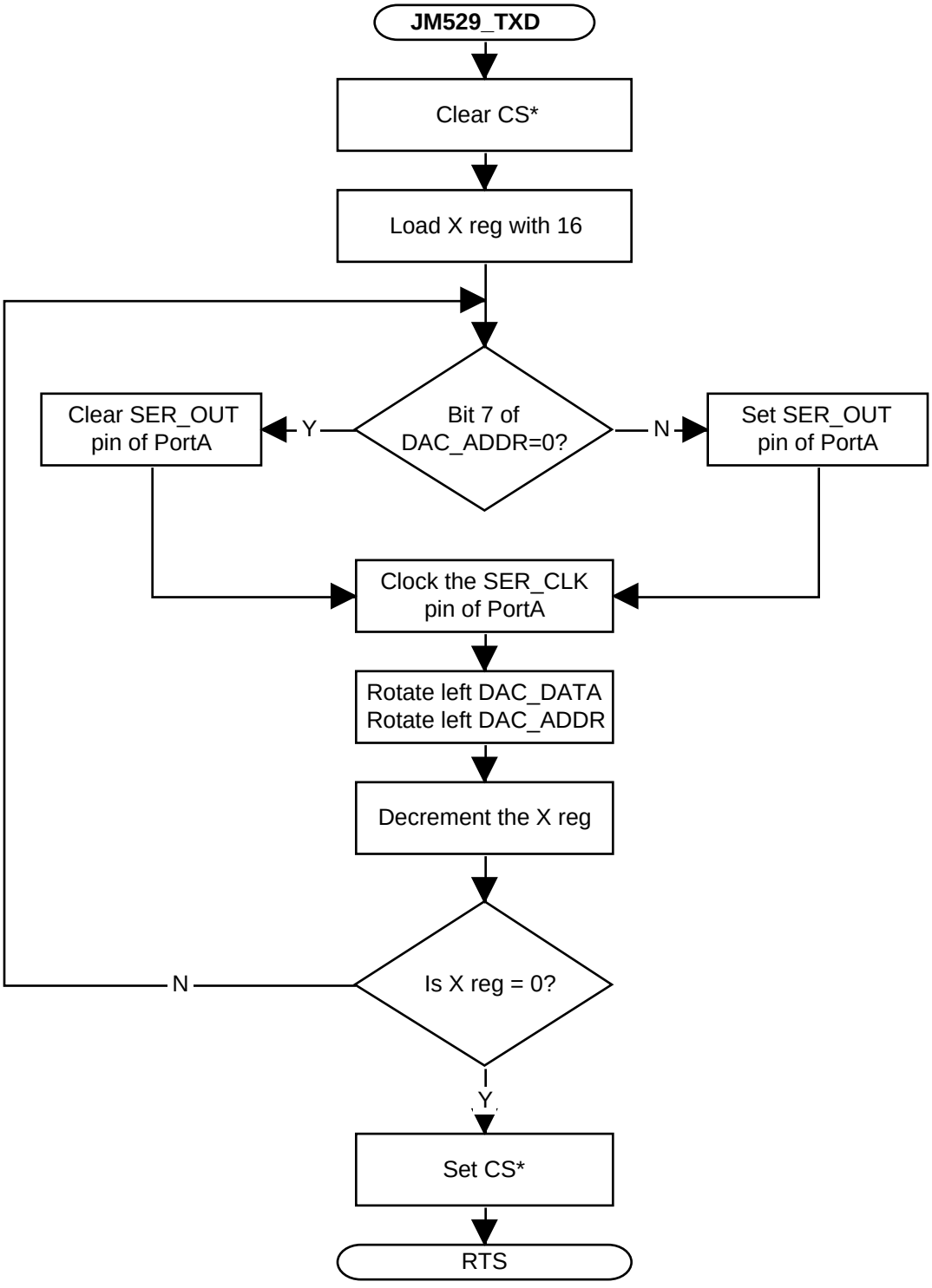
```
*****************************************************************************
*                                                                           *
*       CM529_TXD SubRoutine                                                *
*                                                                           *
* This subroutine will write the address and data info to the MAX529        *
*                                                                           *
* Conditions: DAC_ADDR and DAC_DATA defined                                 *
* Destroys: X, ACCA                                                         *
*                                                                           *
*****************************************************************************

*       Send out 16 bit frame   *
CM529_TXD       bclr    CS,PORTA        ;CS* is low

*       Send out address
                lda     DAC_ADDR        ;load ACCA with ADDR
                sta     SPDR            ;store ACCA to spi data reg
CM529_W1        brclr   7,SPSR,CM529_W1 ;wait until SPIF flag is set

*       Send out data
                lda     DAC_DATA        ;load ACCA with DATA
                sta     SPDR            ;store ACCA to spi data reg
CM529_W2        brclr   7,SPSR,CM529_W2 ;wait until SPIF flag is set

                bset    CS,PORTA        ;CS* is high, end 16 bit frame
                rts                     ;return from subroutine

*****************************************************************************
*                                                                           *
*       Interrupt and Reset vectors for Main Routine                        *
*                                                                           *
*****************************************************************************

                org     RESET
                fdb     CM529_START
```

Appendix D


HC705J1A to MAX529 Interface
Motorola - CSIC Strategic Applications
Title
HC705J1A to MAX529
Size
A
Document Number
705J1A.SCH
REV
R10
Date:
April 9, 1995
Sheet
1 of 1
VDDA
R1
1.5K
R2
1.0K
AGND
U3
MAX529
REFH1
REFL1
REFH2
REFL2
VDD
GND
VSS
CS
CLK
DIN
DOUT
SHDN
OUT0
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
C2
0.1uF
CHIP SELECT
SERIAL_CLOCK
SERIAL_OUT
U1
MC68HC705J1AP
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
OSC1
OSC2
RESET
IRQ/VPP
PB0
PB1
PB2
PB3
PB4
PB5
X1
4.0MHz
C1
0.1uF
U2
MC34064
INPUT
RESET
GND

Appendix E
HC705J1A/529 Flowchart




JM529_TXD
Clear CS*
Load X reg with 16
Bit 7 of DAC_ADDR=0?
Y
N
Clear SER_OUT pin of PortA
Set SER_OUT pin of PortA
Clock the SER_CLK pin of PortA
Rotate left DAC_DATA
Rotate left DAC_ADDR
Decrement the X reg
Is X reg = 0?
N
Y
Set CS*
RTS

Appendix F
HC705J1A/529 Assembly Code

```
*****************************************************************************
*****************************************************************************
*                                                                           *
*            Main Routine J1A_M529 - 705J1A to MAXIM MAX529 DAC             *
*                                                                           *
*****************************************************************************
*                                                                           *
* File Name: J1A_M529.RTN                    Copyright (c) Motorola 1995    *
*                                                                           *
* Full Functional Description Of Routine Design:                            *
*   Program flow:                                                           *
*       Reset:     Initializes ports for bit banging.                       *
*                  Set up MAX529 for half-buffered mode                     *
*                  Initialize DAC address and count for test                *
*                  Execute continuous loop to create stair step function    *
*       JM529_TXD: Clear CS                                                 *
*                  Loop 16 times                                            *
*                    Write address and data on port pin clock it            *
*                  Loop done                                                *
*                  Set CS                                                   *
*                                                                           *
*****************************************************************************
*                                                                           *
*       Part Specific Framework Includes Section                            *
*                                                                           *
* Place the assembler statement (#INCLUDE) to include the part specific     *
* framework for the target part.                                            *
*                                                                           *
*****************************************************************************

#nolist
#INCLUDE 'H705J1A.FRK'                  ;Include the equates for the HC705J1A
                                        ;so that all labels can be used.
#list

*****************************************************************************
*                                                                           *
*       MOR Bytes Definitions for Main Routine                              *
*                                                                           *
*****************************************************************************

                org     MOR
                db      $21             ;COP enabled, osc resistor enabled
                                        ;If used on a mask rom part,
                                        ; be sure to specify this option.
```

```
*****************************************************************************
*                                                                           *
*       Equates and RAM Storage                                             *
*                                                                           *
*****************************************************************************

CS              equ     0               ;bit # for chip select
SER_CLK         equ     1               ;bit # for serial clock
SER_OUT         equ     2               ;bit # for serial data out

***     RAM storage variables   ***

                org     RAM             ;start of static RAM at $C0
DAC_ADDR        rmb     1               ;1 byte needed for DAC address
DAC_DATA        rmb     1               ;1 byte needed for DAC data
COUNT           rmb     1               ;1 byte for counting

*****************************************************************************
*                                                                           *
*       Program Initialization                                              *
*                                                                           *
* This section sets up the port for bit banging.                            *
*                                                                           *
* To prevent floating inputs and associated high current draw,              *
* the HC705J1A has pulldown devices on all I/O pins. This                   *
* initialization should enable these pulldowns on unused I/O                *
* pins. RESET_ enables the pulldowns, so no code is required.               *
*                                                                           *
*****************************************************************************

                org     EPROM           ;start of user eprom at $300
JM529_START     lda     #$01
                sta     PORTA           ;init portA
                lda     #$07
                sta     DDRA            ;init port A dir

*****************************************************************************
*                                                                           *
*       J1A_M529 Main Program Loop                                          *
*                                                                           *
* The code runs through the routine to check for                            *
* proper serial transmission. Each DAC channel will have a 382 Hz           *
* stair step function                                                       *
*                                                                           *
*****************************************************************************

*       Set up DAC for un-buffered output mode
                lda     #$00            ;load up DAC_ADDR=$00
                sta     DAC_ADDR
                lda     #$80            ;load up DAC_DATA=$80
                sta     DAC_DATA
                jsr     JM529_TXD       ;transmit info to MAX529
```

```
*       Initialize DAC address and COUNT for test
                lda     #$FF            ;all DAC channels will be tested
                sta     DAC_ADDR
                clr     COUNT           ;clear COUNT

*       Loop to output stair step function on all 8 DAC outputs
JM529_Loop      lda     COUNT
                sta     DAC_DATA        ;store COUNT to DAC_DATA
                jsr     JM529_TXD       ;transmit info to MAX529

                lda     COUNT           ;add $20 to COUNT
                add     #$20
                sta     COUNT

*       Kick the WatchDog and loop back
JM529_BRANCH    lda     #$00            ;reset COP
                sta     COPR
                bra     JM529_Loop      ;branch to Loop

*************************************************************************
*                                                                       *
*       JM529_TXD SubRoutine                                            *
*                                                                       *
* This subroutine will write the address and data info to the MAX529    *
*                                                                       *
* Conditions: DAC_ADDR and DAC_DATA defined                             *
* Destroys: X                                                           *
*                                                                       *
*************************************************************************

*       Send out 16 bit frame   *
JM529_TXD       bclr    CS,PORTA        ;CS* is low

                ldx     #16T            ;load X with 16

***     Write the serial output pin
WRITE           brclr   7,DAC_ADDR,JM529_C  ;if temp bit7 = 0,
                                        ; goto jm529_c
                bset    SER_OUT,PORTA   ;ser_out = 1
                bra     JM529_CLOCK     ;goto jm529_clock
JM529_C         bclr    SER_OUT,PORTA   ;ser_out = 0

***     Clock the serial clock pin
JM529_CLOCK     bset    SER_CLK,PORTA   ;ser_clk = 1
                bclr    SER_CLK,PORTA   ;ser_clk = 0
                rol     DAC_DATA        ;rotate left DAC_DATA
                rol     DAC_ADDR        ;rotate left DAC_ADDR

                decx                    ;decrease counter loop
                bne     WRITE           ;is the count finished?

                bset    CS,PORTA        ;CS* is high, end 16 bit frame
                rts
```

```
*****************************************************************************
*                                                                           *
*       Interrupt and Reset vectors for Main Routine                        *
*                                                                           *
*****************************************************************************

                org     RESET
                fdb     JM529_START
```

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

How to reach us:

MFAX: RMFAX0@email.sps.mot.com – TOUCHTONE (602) 244-6609

INTERNET: http://Design-NET.com

USA/EUROPE: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036. 1-800-441-2447

JAPAN: Nippon Motorola Ltd.; Tatsumi-SPD-JLDC, Toshikatsu Otsuki, 6F Seibu-Butsuryu-Center, 3-14-2 Tatsumi Koto-Ku, Tokyo 135, Japan. 03-3521-8315

HONG KONG: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

MOTOROLA

AN1256/D